Kantutjämning i realtidsapplikationer

Andreas Salminen, 38875
Kandidatexamen i datateknik
Handledare: Jan Westerholm
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
2019

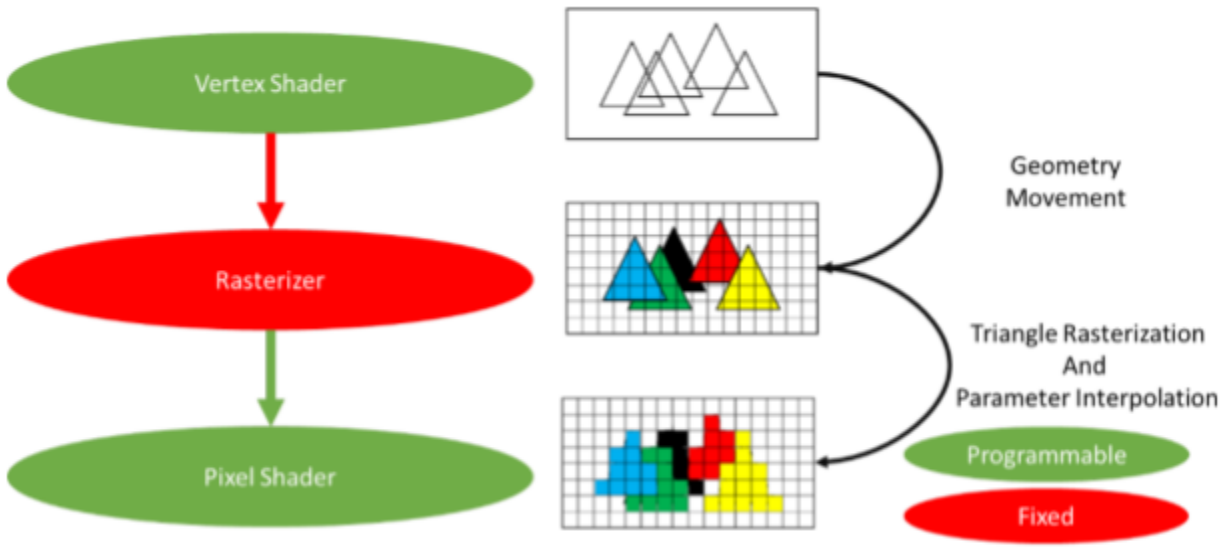
Referat

Avhandlingen behandlar olika kantutjämningsalgoritmer i realtidsapplikationer. Kantutjämning används i grafik för att göra bilder prydligare genom att släta till hackiga kanter. I avhandlingen tas först upp bilder som ritas tvådimensionellt på en datorskärm och varför de innehåller ojämna kanter. Sedan beskrivs några av de vanligaste algoritmerna som används för kantutjämning i bilder. Eftersom de olika algoritmerna skapar kantutjämnade bilder på olika sätt med varierande resultat, blir hastigheten av algoritmen samt den slutliga kvaliteten av bilden avgörande.

Innehållsförteckning

1. Inledning

En bild på skärmen är uppbyggd av pixlar som skapas genom en process som kallas renderingspipa (på engelska rendering pipeline) [5]. Om de geometriska figurerna som ska representeras i 2D på skärmen existerar i en 3D-rymd kommer pixlarna för bilden i 2D att samplas genom en process som kallas rastrering [1]. Rastreringen mappar geometriska figurer till pixlar på skärmen [5]. Bilden som syns på skärmen kommer inte att vara en exakt reproduktion av det som finns i en oändligt exakt matematisk grafikrymd [1]. Detta beror på att grafiken är uppbyggd av oändligt små punkter som sedan översätts till pixlar på skärmen. Varje pixel på skärmen blir alltså en approximering av informationspunkterna närmast pixeln [5]. Eftersom olika föremål i en bild kan ha stora skillnader i färg kommer föremålen i resultatet att innehålla ojämna kanter. Detta kallas rumslig aliasing (på engelska spatial aliasing). En annan typ av aliasing är temporal aliasing, vilket innebär att tunna objekt hamnar mellan två pixlar (på engelska pixel flickering) [5].



Figur 1. Renderingspipan. Färgerna kommer till i bild 3, men syns i bild 2 för att urskilja trianglarna [6].

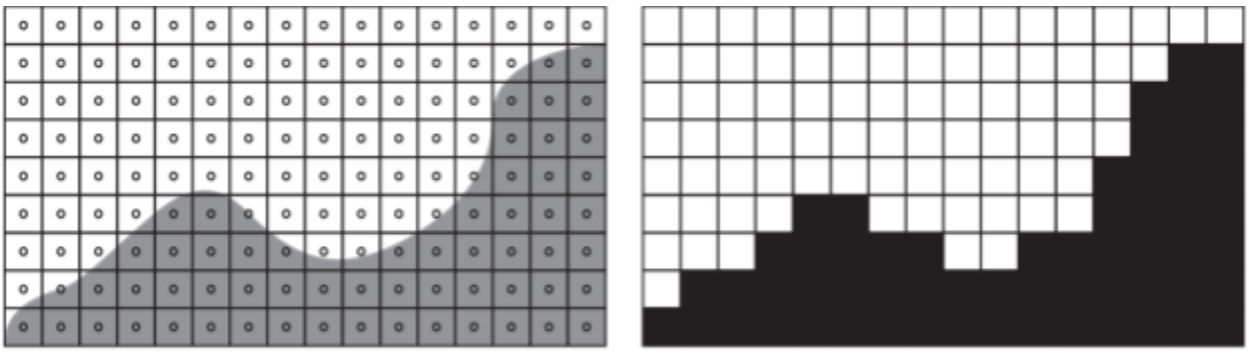
Ett sätt att förminska ojämnheter som uppstår är att öka resolutionen på skärmen som visar den fullständiga 2-dimensionella bilden. Ökningen av antalet pixlar gör att hackiga kanter blir märkbart mindre synliga, eftersom antalet samplingspunkter också ökar. Nackdelen med att öka resolutionen är en ökad belastning av hårdvaran [3], vilket gör det långsammare för grafikkortet att skapa bilder. Tiden det tar för datorn att skapa en bild ökar proportionellt med antalet pixlar som samplas. Ökningen av resolution fungerar så länge som antalet bilder som skapas per sekund hålls på en acceptabel nivå. Ett vanligt mål för realtidsapplikationer som till exempel spel är 60 bilder per sekund.

Eftersom bildkvalitét är subjektivt kommer resultaten från algoritmerna inte att direkt avslöja vilken som är bäst för alla. Det går dock att skapa en överblick av de olika algoritmernas fördelar och nackdelar, vilket hjälper till då man behöver välja mellan de olika algoritmerna. Hastigheten mellan algoritmerna varierar och är inte direkt proportionell till upplevelsen.

Det går att dela in algoritmerna i två grupper: Nedsamplings algoritmer och Post-processing algoritmer. Till nedsamplings algoritmer hör de algoritmer som körs före en tvådimensionell bild av scenen är skapad. Nedsamplings algoritmer använder sig av flera samplingspunkter för att skapa bilder. Ett sampel är en mätning av en exakt punkt i en 3D-rymd. Till post-processing algoritmer hör de algoritmer som suddar till kanter i en färdig tvådimensionell bild av scenen. Post processing algoritmer måste köras före eventuell text ritas på skärmen för att undvika oönskad kantutjämning av texten. Nedsamplings algoritmer har tillgång till en större mängd data och skapar därför vanligtvis bilder av högre kvalitét, men användningsmöjligheterna beror mycket på hur hög förlusten av prestandan är jämfört med post-processing algoritmer.

För att få ett bra resultat är det viktigt för en algoritm att veta var kanterna av saker finns på en bild. Annars kommer bilden att lätt bli antingen för skarp på grund av för lite kantutjämning, eller suddig på grund av för mycket kantutjämning [3].

Bilderna nedan i fig.2 är delade i rutor. Varje ruta representerar en pixel på en datorskärm. Varje pixel i datorskärmen måste färgas helt, vilket betyder att pixlarna aldrig kan vara delvis på eller av [2]. Bilden till vänster representerar verkligheten och kunde teoretiskt representeras på en datorskärm om den hade oändligt hög pixeldensitet. Bilden till höger skapas genom att sampla bilden till vänster. Samplingen sker i mitten av varje pixel. För varje samplingspunkt som ligger på insidan av figuren färgas motsvarande pixel. Bilden till höger visar hur bilden till vänster kommer att se ut då den representeras med hjälp av pixlar på en vanlig datorskärm. Bilden blir kantig och ser inte verklighetstrogen ut. Med kantutjämning försöker man skapa bilder som liknar bilden till vänster utan att öka resolutionen.



Figur 2. Bilden till vänster visar en bild som samplas vid punkterna i mitten. Bilden till höger är resultatet av samplingen.

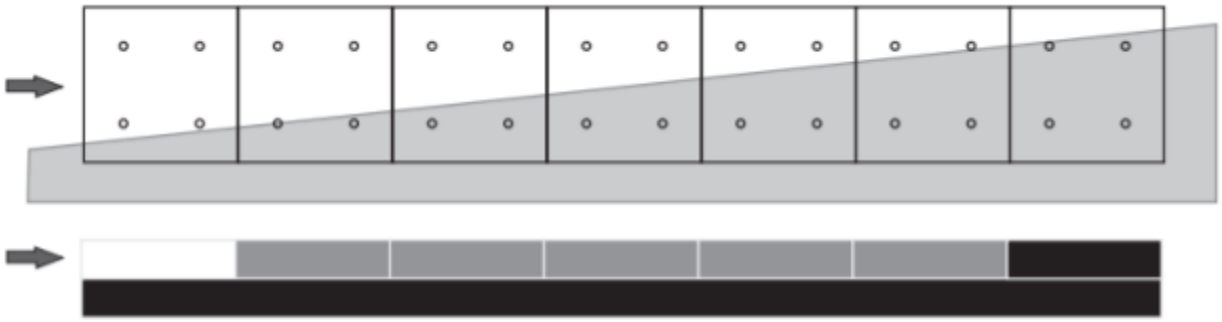
2. Algoritmer för kantutjämning

2.1. Nedsamplings algoritmer

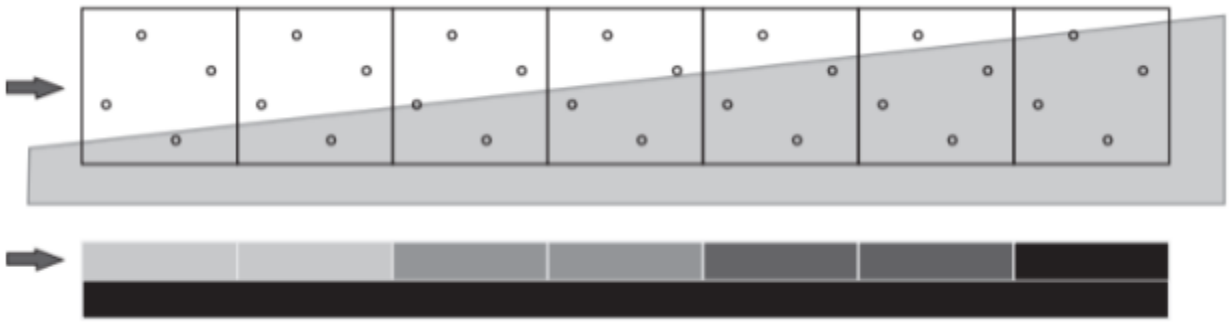
2.1.1. SSAA - supersampling anti-aliasing

Då supersampling anti-aliasing används antar programvaran att den tillgängliga skärmytan är större än den egentligen är [2]. På basis av detta antagande samplas en bild som är större än resolutionen på skärmen. Bilden förminskas sedan till skärmens resolution. Varje pixel som ingår i den bilden som representeras på skärmen har då fått sin färg utifrån en en jämförelse av pixlarna som samplats. Denna metod av anti-aliasing producerar bilder av hög kvalitét, men är mycket långsam på grund av antalet pixlar som samplas [2].

SSAA kan delas in vidare i två grupper: OGSS och RGSS. OGSS står för ordered grid supersampling och RGSS står för rotated grid supersampling. OGSS är den vanligare av de två [2]. RGSS är utan tvekan bättre än OGSS i de flesta fall [2].



Figur 3. Figuren demonstrerar OGSS. I figuren tas det fyra sampel per pixel. För varje sampel som finns inuti föremålet ökar färgens intensitet.



Figur 4. Figuren demonstrerar RGSS. I figuren tas det fyra sampel per pixel. För varje sampel som finns inuti föremålet ökar färgens intensitet.

2.1.2. MSAA - multisample anti-aliasing

Multisample anti-aliasing baserar sig på SSAA, men är en mera optimerad version av den. MSAA-algoritmen samplar två eller flera pixlar som befinner sig intill varandra istället för att sampla hela scenen. Algoritmen vinner tid genom att behandla klumpar av pixlar med liknande färger som en helhet, istället för att behandla pixlarna skilt för sig. MSAA räknar färgen för en pixel endast en gång [3]. Då SSAA-algoritmen används betyder SSAAx4 att en pixel samplas fyra gånger, medan MSAAx4 ibland tillåter att samplingen delas mellan intilliggande pixlar. Optimeringarna som MSAA-algoritmen utför kommer ibland att försämra bildkvalitéten en aning jämfört med SSAA.

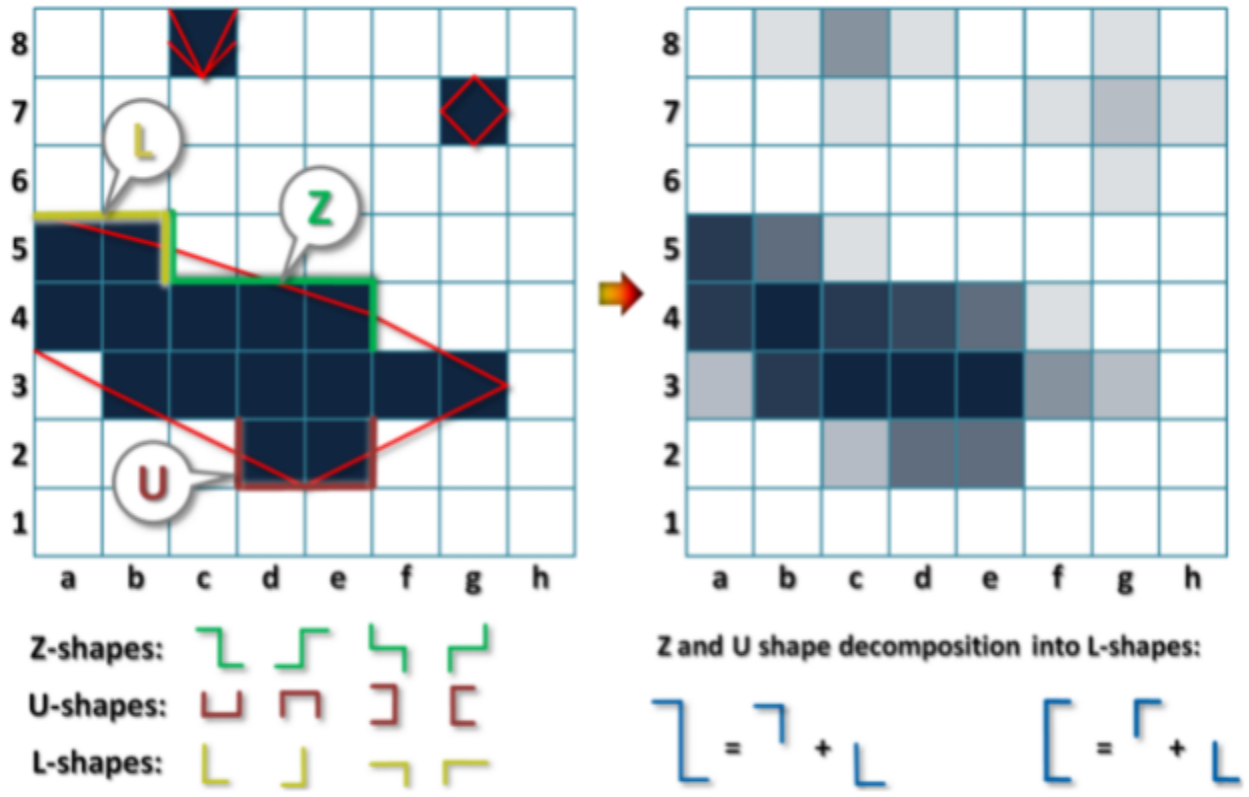
Den största nackdelen med MSAA är att skuggningsprocessen (på engelska deferred shading) inte är kompatibel med MSAA [5]. MSAA blir snabbt långsamt vid uppskalning och använder mycket minne, och kan därför vara ett sämre val då det finns ett behov av högre hastigheter [3]. Med MSAAx8 går det endast att skapa färggradienter med åtta olika färger längs en kant [4].

2.2. Post-processing algoritmer

2.2.1. MLAA - Morphological Anti-aliasing

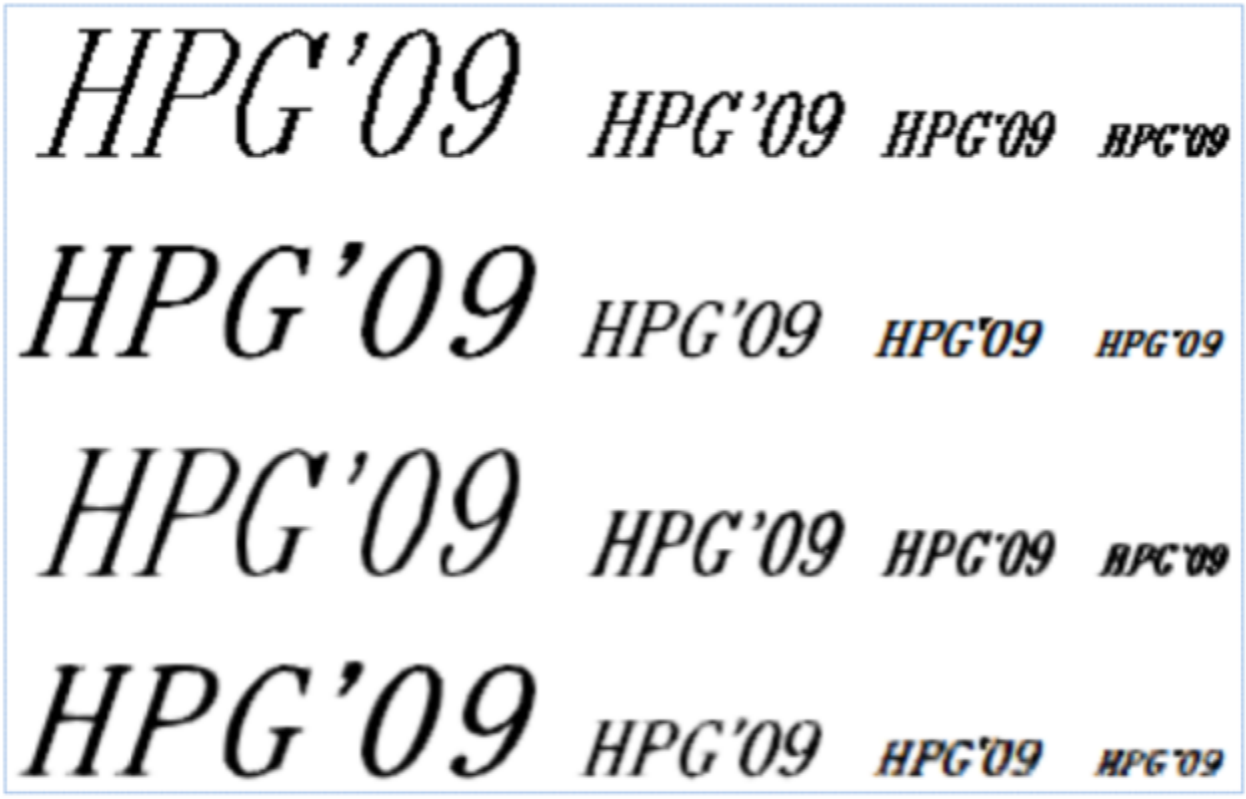
Intels algoritm morphological anti-aliasing (på svenska morfologisk kantutjämning) är en av de första post-processing algoritmerna för kantutjämning. MLAA tar i beaktande hur en människa urskiljer mönster i bilder och fokuserar kantutjämningen vid dem [7]. Idén med MLAA är att återskapa förlorad information som uppstår då bilder skapas [5].

MLAA har tre steg. Det första som händer är att algoritmen identifierar U-, L- och Z-formade mönster med märkbara skillnader. Pixlar som identifieras som märkbart olika i färg bildar linjer. Eftersom dessa linjers färger har märkbara skillnader kommer de att synas bra och bilda tydliga kanter. I det andra steget skapas en geometrisk form på basis av linjerna i steg ett. Den nya formen består av nya linjer som inte behöver vara antingen vertikala eller horisontella. I det tredje steget skapas en ny bild genom att jämföra färgerna av alla former som skapades i steg två med pixlarna intill dem. Den nya bilden kommer att innehålla former där kanternas färg är ett medeltal av färgerna som fanns förut och färgerna från kanterna bredvid dem.



Figur 5. Bilden till vänster innehåller originalbilden med de identifierade L-, Z- och U-formerna. De röda strecken i bilden föreställer formen som skapas i steg två av algoritmen. Bilden till höger är resultatet av kantutjämningen.

MLAA är en mycket snabb algoritm och använder sig endast av färger i bilden. MLAA mättes vara ungefär 10 gånger snabbare än MSAA på ett GTX-9800 grafikkort [4]. Algoritmen använder sig av endast ett sampel per pixel. Resultatet av MLAA är jämförbart med SSAA [7]. Att bearbeta små detaljer i bilder som endast består av en pixel eller är mycket tunna är dock inte möjligt för MLAA, vilket är algoritmens största nackdel [7]. En del figurer förlorar sin form vid hörnen, eftersom de kan bli rundade [3]. MLAA kan inte urskilja text vilket kan leda till svårläst eller helt oläsbar text. I synnerhet text som redan behandlats med någon kantutjämning kommer att bli extremt suddigt [7].



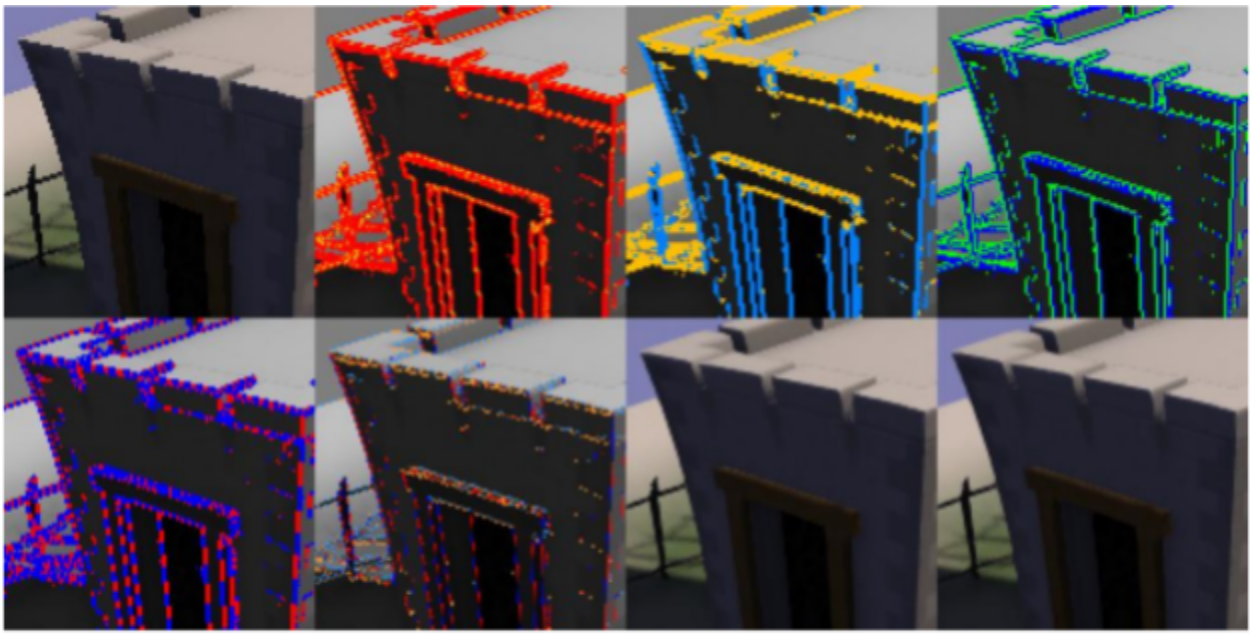
Figur 6. Rad 1: bitmap font utan kantutjämning. Rad 2: TrueType font med kantutjämning. Rad 3: bitmap font med MLAA. Rad 4: TrueType font med kantutjämning samt MLAA. Fontstorlekar: 24, 12, 8 och 6.

2.2.2. FXAA - fast approximate anti-aliasing

Nvidias algoritm FXAA är en avancerad kantutjämningsmetod skapad av Timothy Lottes [4]. Algoritmen identifierar hackiga kanter på liknande sätt som MLAA [4]. FXAA klarar av att identifiera och kantutjämna enstaka pixlar samt subpixlar [6].

Algoritmen har som input en bild med RGB data som används till att bestämma ljusstyrkan för pixlarna. Till näst identifieras kontrasten mellan pixlar för att få reda på var kanter finns. I fig. 7 är kanterna som identifierats markerade med rött. Den gula färgen står för subpixel områden som behöver kantutjämning. I den tredje bilden i fig. 7

är pixlarna som algoritmen identifierat som kanter markerade som horisontella eller vertikala. De horisontella är markerade i guld och de vertikala är markerade i blått. På basis av riktningen av kanterna markeras de pixelparen som är vinkelrätt mot kanten med högsta kontrast. Paren är markerade i den fjärde bilden i fig. 7. Slutet av kanterna söks upp längs kanterna i både negativ riktning och positiv riktning. Riktningarna är markerade i den femte bilden i rött och blått. Algoritmen söker efter större förändringar i genomsnittlig luminans mellan de pixlelparen som identifierats i föregående steg. På basis av kanternas slut transformeras pixlarnas position vid kanterna vinkelrätt. Den sjätte bilden demonstrerar detta med rött och blått för horisontell flyttning och guld och ljusblå för vertikal flyttning av pixlar. I den sjunde bilden syns den nya omsamplade bilden som samplats med de nya värdena för pixlarna. Slutligen i bild åtta har ett lågpassfilter körts på basis av antalet subpixel kantutjämning som behövs. [6]



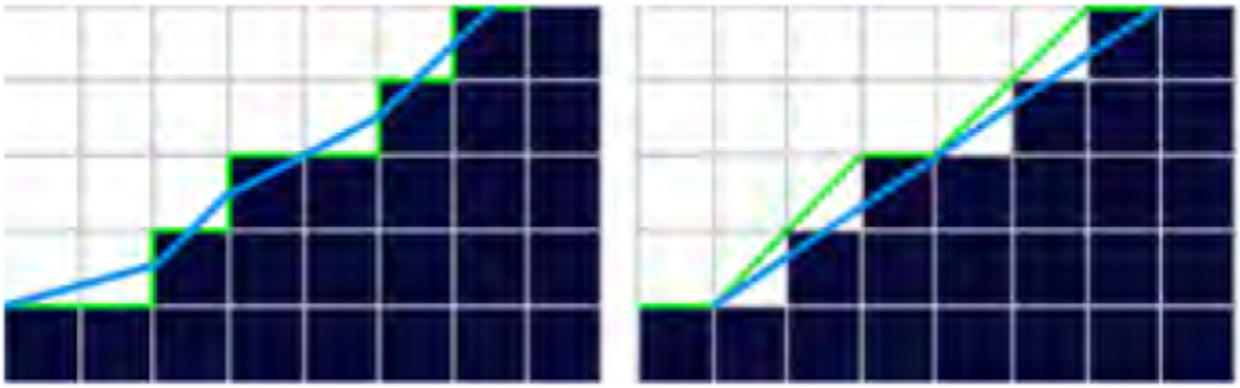
Figur 7. FXAA algoritmen steg för steg

FXAA är en mycket snabb algoritm som på ett GTX-480 grafikkort klarar av att producera kantutjämning av en bild av resolutionen1920x1200 på <1 ms [6]. Nackdelar med FXAA är att texturer inuti figurer också kommer att bli suddigare, vilket ibland är oönskat.

2.2.3. SMAA - enhanced subpixel morphological anti-aliasing

SMAA är en vidareutveckling av MLAA [3]. Algoritmens delsteg är förbättrade eller helt förnyade från MLAA [3]. Tanken med SMAA är att försöka lösa problem som är vanliga i post-processing metoder [3]. De flesta post-processing metoder tar endast i beaktande hur pixlar skiljer sig från varandra med numeriska värden, utan att ta i beaktande hur en människa skulle se kanter i en form [3]. Ett objekt kan få för rundade kanter och därför förlora sin originalform, vilket syns bra i text och skarpa hörn. De flesta metoderna behandlar endast horisontella och vertikala mönster utan att ta i beaktande diagonala mönster [3].

SMAA löser problemen som uppstår med post-processing algoritmer med hjälp av en modulär lösning. Varje problem behandlas skilt för sig. Algoritmen identifierar först ett större antal mönster, samt flera olika typer av mönster än MLAA. De nya mönstren som inte MLAA använder sig av inkluderar diagonala mönster [3]. I figur 8 jämförs de nya diagonala linjernas effekt med MLAA. De gröna linjerna i figuren visar de nya diagonala linjerna som SMAA använder. I blått är de nya linjerna som skapas med hjälp av de gröna. Resultatet av SMAA blir närmare originalet än MLAA [3].

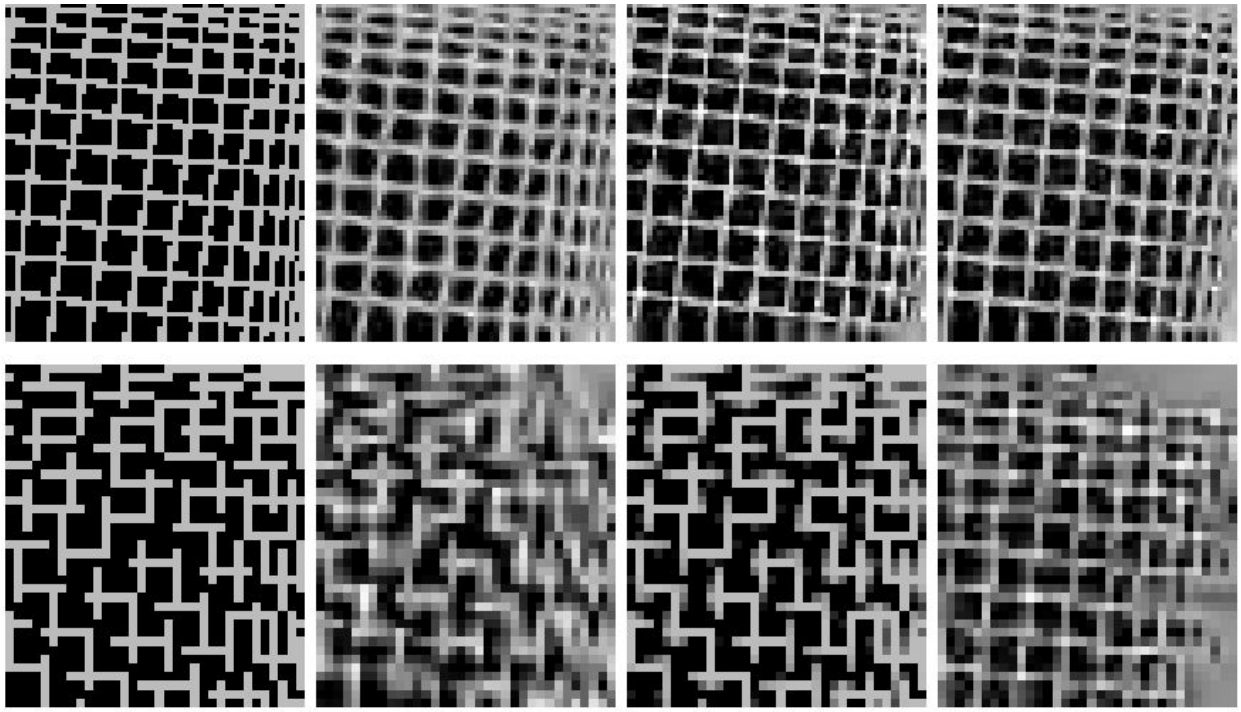


Figur 8. Jämförelse av diagonalerna mellan MLAA och SMAA. Till vänster är MLAA och till höger är SMAA.

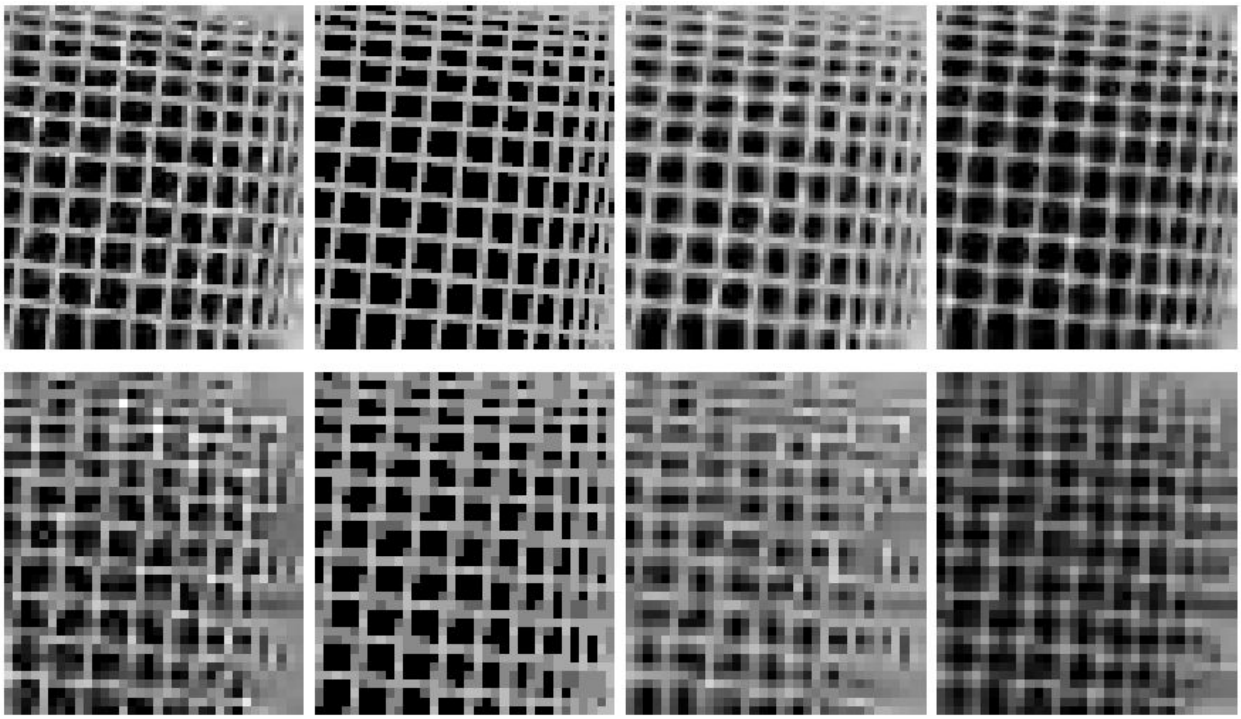
Eftersom SMAA är modulärt är det möjligt att aktivera och inaktivera olika delar av algoritmen. Denna funktionalitet hjälper till om man behöver skräddarsy algoritmen så att den uppfyller grafiska krav eller fungerar på den hårdvara som finns tillgängligt [3].

Endast övre och vänstra sidan av varje pixel analyseras då kanterna identifieras, vilket sparar tid. De andra sidorna fås ur intilliggande pixlar [3]. Då SMAA kördes på en GTX-470 mättes hastigheten att kantutjämna en 1080p bild. I medeltal tog det 1.3ms med SMAA T2x och 2.6ms med SMAA T4x [3].

Ett lutande vitt rutfält över en svart bakgrund syns i figurerna 9 och 10. Figurerna visar hur detta svåra fall för kantutjämnignsmetoderna MLAA och SMAA 1x ser ut [3]. Problemet är att de vita rutorna förlorar sin kontinuitet, vilket hindrar återskapningen av dem [3]. De nedre bilderna är från längre avstånd inzoomade versioner av de övre. SMAA S2x och SMAA 4x närmar sig i kvalitéten av MSAA [3]. Konfigurationerna SMAA 1x, SMAA S2x, SMAA T2x och SMAA 4x skiljer sig från varandra enligt följande: SMAA S2x innehåller allting som SMAA 1x innehåller, men också spatial multisampling. SMAA T2x fungerar som SMAA S2x förutom att den istället för spatial multisampling använder temporal supersampling. SMAA 4x innehåller både spatial multisampling och temporal supersampling [3].



Figur 9.



Figur 10.

2.2.4. TAA - temporal anti-aliasing

Då rastreringen sker i renderingspipan uppstår det fel som TAA rättar till [5]. Principen med TAA är att använda sig av föregående bilder tillsammans med bilden som renderas som bäst [5]. Olika varianter av TAA är till exempel TXAA och TRAA. TXAA är Nvidias version av TAA.

TRAA står för Temporal Reprojection Anti-Aliasing. Algoritmen fungerar genom att spara flera föregående bilder i en buffert, som sedan blandas med bilden som skapas som bäst. Detta kallas på engelska reprojection och sker genom att söka upp färger från föregående bilder för att skapa den nuvarande bilden [5]. För att TRAA ska fungera behövs det en del andra tekniker också [5]. Kamera darrning (på engelska camera jittering) är en teknik som behövs för att återskapa pixlar runt kanter. En velocity buffer, som på svenska direkt översatt blir hastighetsbuffert används till att bestämma positionen av pixlar på föregående bild om de har flyttat på sig. En bildbuffert (på engelska frame history buffer) behövs för att spara föregående bilder som sedan används för skapandet av den nuvarande bilden. En “låda” (på engelska clipping color box) används till att begränsa bildbufferten så att inte brus eller felaktiga färger hamnar dit. Ett skärpningsfilter (på engelska sharpen filter) hjälper till att minska suddningseffekten som sker. För att korrigera för objekt som flyttar sig för snabbt för clipping color box behövs slutligen ännu suddning för rörelser (på engelska motion blur).

3. Slutsatser

Visandet av bilder är ofta flaskhalsat av skärmens uppdateringsfrekvens, vilket betyder att en del bilder som skapas av grafikkortet aldrig hinner visas på skärmen ifall grafikkortet klarar av att skapa flera bilder än skärmen kan visa. Om grafikkortet klarar av att skapa betydligt högre antal bilder än skärmen kan visa kan det löna sig att välja en sådan kantutjämningsmetod som inte snålar på resursanvändningen. MSAAx8 ger mycket bra bildkvalitét i sådana fall där bildfrekvensen inte blir ett problem.

Realtidsapplikationer som körs på förmånlig (långsam) hårdvara måste ofta samplas i låg resolution för att behålla målet av 60 bilder per sekund. Lyckligtvis klarar de snabba post-processing algoritmerna av att producera bilder som i de flesta fall är jämförbara med nedsamplingsalgoritmer. FXAA och SMAA är bra postprocessing algoritmer som det lönar sig att prova om de finns tillgängliga i applikationen som körs. TAA anses ofta sudda till bilden för mycket, men kan ändå vara värd att prova. Nedsamplingsalgoritmer kan också fungera på långsam hårdvara om valet av antalet sampel är möjligt.

4. Litteraturförteckning

[1] D. Hearn, M. Pauline Baker, and W. Carithers, Computer Graphics with OpenGL. Pearson Education, 2011.

[2] K. Beets, D. Barron, “Super-sampling Anti-aliasing Analyzed”, Beyond3D, 2000, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.1915

[3] J. Jimenez, J. I. Echevarria, T. Sousa, and D. Gutierrez, “SMAA: Enhanced Subpixel Morphological Antialiasing,” Comput. Graph. Forum, vol. 31, no. 2pt1, pp. 355–364, 2012.

[4] J. Jimenez et al., “Filtering approaches for real-time anti-aliasing,” ACM SIGGRAPH 2011 Courses on - SIGGRAPH ’11, 2011, http://dx.doi.org/10.1145/2037636.2037642

[5] C. A. O. Labrador, “Improved Sampling for Temporal Anti-Aliasing,” Department of Computer Science, Lund University, 2018, http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8971248&fileOId=8971249

[6] T. Lottes, “FXAA,” Nvidia, 2009,
https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf

[7] A. Reshetov, “Morphological antialiasing,” in Proceedings of the 1st ACM conference on High Performance Graphics - HPG ’09, 2009, http://dx.doi.org/10.1145/1572769.1572787

5. Bildförteckning

Fig. 1 C. A. O. Labrador, “Improved Sampling for Temporal Anti-Aliasing,” Department of Computer Science, Lund University, 2018, http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8971248&fileOId=8971249

Fig. 2, 3, 4 K. Beets, D. Barron, “Super-sampling Anti-aliasing Analyzed”, Beyond3D, 2000, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.1915

Fig. 5, 6 A. Reshetov, “Morphological antialiasing,” in Proceedings of the 1st ACM conference on High Performance Graphics - HPG ’09, 2009, http://dx.doi.org/10.1145/1572769.1572787

Fig. 7 T. Lottes, “FXAA,” Nvidia, 2009, https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf

Fig. 8, 9, 10 J. Jimenez, J. I. Echevarria, T. Sousa, and D. Gutierrez, “SMAA: Enhanced Subpixel Morphological Antialiasing,” Comput. Graph. Forum, vol. 31, no. 2pt1, pp. 355–364, 2012